Exercise 1

WS 2020 - 188.977 Grundlagen des Information Retrieval

Organization

Deadline: 16.12, 23:55 upload in TUWEL

2 persons per group

Exercise will be evaluated based on your submitted report and code

In case of questions, use the TUWEL forum so that others also benefit from the question/answer. Alternatively, you might also contact me markus.zlabinger@tuwien.ac.at directly.

Use the programming language Python3

Goal (Total of 100 points)

The main goal of this exercise is to create, evaluate, and describe your own search engine. To test it, we use a subset of Wikipedia articles. More details are in the notes below. The individual goals are the following:

- Implement efficient text processing code to create an inverted index from the given Wikipedia articles, including: tokenization, lowercasing, stemming, stop words **(40 points for evaluation-set, (only 10 points for dev-set))**

- Implement save/load to disk code for your inverted index, including a format to save the index **(5 points)**

- Implement query processing: (re-use tokenization, stemming) including: TF-IDF and BM25 scoring methods **(20 points)**
  - Exploration mode: allow free text input and return ranked documents in full text
  - Evaluation mode: read a topic file and output ranked document ids in the evaluation format

- Evaluate the given INEX Wiki test collection **(10 points only for evaluation-set)**
  - Report MAP, NDCG@10, Precision@10, Recall@10 for TF-IDF and BM25 configurations (using trec_eval)

- Write a report: **(25 points)**
  - Describe your architecture briefly
  - Note 2 or more interesting details of your work (coming up with the interesting things is part of the exercise)
  - List your performance numbers (indexing time, query time for evaluation queries)
  - List the evaluation results + observations between the two scoring methods

Bonus points

Do something extra - that we haven't thought of - but it was interesting to you **(up to 10 points)**
Note: this extra thing can be some extraordinary efficient code (including, but not limited to: SIMD, GPU acceleration) or some visual analysis of the index, results, etc.. or some inspection of the test collection… Or implement additional features… Describe your attempt to obtain the bonus points in the report.

Notes

- Hand-in: Upload your (i) code, (ii) report (as PDF), and (iii) results in TREC format in TUWEL.
- Use the provided template structure for the source code (folder *code*), results (folder *retrieval_results*) and report (folder *report*). You can find the template in TUWEL.
- You are free to choose which architecture and methods you like best to achieve the goals (e.g. type of inverted index creation process, how you structure your data structures, etc..)
- Regarding the use of libraries: The idea of this exercise is to write your own text-processing and indexing code. So that means in addition to low-level methods (such as encoding, file i/o, string methods, list, hash-tables, trees, memory management helpers, etc..) only a stemming library is allowed. Therefore, libraries for tokenization, automatic tf-idf generation, stop word remove methods and so on are not allowed. You can import an NLP or search engine library to get an array of stop words and the stemming class, but not more. Additionally, you may use an XML parser for the parsing of multiple articles per file - but the XML structure is very basic and can easily done with your own code.
- You don't have to store the document text for the exploration mode, just save the original filename + start/end bytes and load the text. However, you might want to consider saving the preprocessed text (how it is added to the inverted index) per document to explore your tokenization etc.. which gives you an accurate representation of your indexed data.
- You don't have to create a graphic user interface, just use a console based app, that can switch between the exploration and evaluation mode and between TFIDF and BM25.
- Feel free to look at code samples, search engine implementations, algorithm descriptions - but do not copy code. If you find some good references and ideas add them to the report.
- Memory management: The evaluation corpus contains slightly more than 2 GB of documents. If you index these documents using standard Python data types (e.g. int()), you will quickly drain your memory. We recommend that you store the posting lists as numpy arrays, where you can define data types for an efficient storage of your posing list. To illustrate the advantage in using numpy arrays, consider the following code example:

  ```
  $ var = np.array([1,2,3,4,5], dtype=np.uint32) # requires 20 bytes of memory
  ```

  ```
  $ var = [1,2,3,4,5] # requires 140 bytes of memory
  ```

- Describe your prototype using the provided template. Describe briefly your index and what information is stored at indexing time. Finally, report your evaluation results calculated using *trec_eval*. **Try to find a justification of why some of your retrieval models performed better than the others.** The report must explain how to run the prototype. Maximum size: 4 pages or 2000 words.

Information on dataset & evaluation

Dataset:

- 2 folders with Wikipedia articles (dev-set + evaluation-set)
    - Both contain files with multiple randomly sorted Wikipedia articles in xml format (the body of the articles only contain plaintext)
    - Dev-set: contains only relevant documents, you may use it to develop your implementation and evaluation pipeline, but not for the actual evaluation
    - Evaluation-set: contains the dev-set and additional documents (total of 2.2 GB)
- "topics" file - with given evaluation queries and their description
- "qrel" file - which contains human relevance judgements for queries in the topics file = allows us to measure the performance of a search engine and compare different configurations (like TF-IDF and BM25)

Evaluation:

1. Index the articles of a given dataset (e.g. dev-set or evaluation-set)
2. Parse the topics file and get query id and query string (you may choose the title only or also text from the description or narrative)
3. For every query, search in your index for relevant documents and output the top-100 documents per topic in the following format (per line):

   ```
   {topic-id}  Q0  {document-id}  {rank}  {score}  {run-name}
   ```

   - **topic-id** is the id per query from the topics file
   - **document-id** is an identifier for the Wikipedia article
   - **Q0** is just a legacy hardcoded string
   - **rank** is an integer indicating the rank of the doc in the sorted list (normally, this should be ascending from the first line to the last line). Rank starts at 1 (not at 0)
   - **score** the similarity score calculated by the BM25 or TF-IDF method
   - **run-name** a name you give to your experiment (free for you to choose)
4. Use the trec_eval utility (https://github.com/usnistgov/trec_eval) to compute the performance metrics with the following command:

   ```
   $ trec_eval -m map -m ndcg_cut.10 -m P.10 -m recall.10 path/to/qrel_file
   path/to/output_from_3
   ```

   The trec_eval utility has to be compiled with make (on windows: use the Linux subsystem (e.g. bash/ubuntu) for windows)

   Put the result in a text file in the "retrieval_results/" folder in your code repository (the name should roughly describe the configuration like: tfidf_title_only.txt)

References

Wikipedia evaluation collection from: https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/software/inex/

Queries and Ratings (Adhoc Track 2010 Topics) from: https://inex.mmci.uni-saarland.de/data/documentcollection.html